САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра технологии программирования

Гуров Андрей Сергеевич

Выпускная квалификационная работа бакалавра

Разработка мобильного приложения для ОС Android. Специфика проектирования пользовательского интерфейса и клиент-серверной архитектуры

Направление 010400

Прикладная математика и информатика

Научный руководитель,

ст. преподаватель,

Андреев Д. В.

Санкт-Петербург
2017

Оглавление

Введение

В современном мире игры представляют собой не только средство развлечения, но и объект многочисленных исследований, а их создание — превосходный способ отработки на практике знаний о программировании.

К примеру, среди последних крупных научных статей связанных с компьютерными играми:

1. «Я не буду сажать яблоню, если мир завтра исчезнет» - исследование поведения игроков в кооперативной онлайн игре накануне полного стирания данных.
2. Использование высокодетализированного виртуального мира игры GTA V для отладки алгоритмов компьютерного зрения беспилотных автомобилей.

Согласно последним данным около 63% населения Земли имеет мобильные устройства, из которых более 80% - на базе операционной системы android. Поэтому навыки разработки приложений для этой операционной системы крайне востребованы в современном мире. И помимо непосредственно навыков программирования, для выпуска конечного продукта, необходимо учитывать огромное разнообразие технических и физических характеристик устройств, уметь протестировать устройство, знать особенности взаимодействия разработчика с Google play.

Постановка задачи

Изучить возможности операционной системы Android в следующих областях:

1. Многопоточность
2. Построение пользовательского интерфейса
3. Хранение данных

Полученные в результате знания использовать для создания игры ArtQuest, в которой главная задача игрока: определить художника той или иной картины. Для её создания необходимо реализовать следующий функционал:

1. Запрос, обработка и хранение текстовых данных с сервера
2. Алгоритм подбора вопроса на сервере — рейтинговая система
3. Загрузка и сохранение изображения картины
4. Отображения отдельно взятого вопроса

Глава 1. Структура приложения для операционной системы Android

1.1 Создание приложения. Минимальная версия операционной системы.

Для создания проекта используется официальный инструментарий, предоставляемый компанией Google. Для начала работы, необходимо определиться с API level – версией рабочего окружения программных библиотек (framework API revision), которое могут использовать приложения для взаимодействия с операционной системой. Эти программные библиотеки состоят из:

- Базового набора пакетов и классов
- Набора элементов .xml и атрибутов для декларирования в файле манифеста (manifest file)
- Набора элементов .xml и атрибутов для декларирования и доступа к ресурсам.
- Набора намерений (Intents)
- Набора разрешений доступа к ресурсам устройства и системы, которые приложение может запросить, а также защита ограничения доступа, встроенная в систему

Обновления этих библиотек разработаны с учётом совместимости со всеми предыдущими версиями, иначе говоря — большинство изменений добавочные. Старые версии обновленных компонентов объявляются устаревшими и deprecated – нежелательными к использованию.

Framework API revision указывается в виде целого числа, что позволяет определить минимальные требования к версии установленной на пользовательском устройстве операционной системе.

Как правило, каждому значительному обновлению операционной системы соответствует одно такое число. К примеру, последней вышедшей версии Android 7.1 с кодовым именем Nougat соответствует API level 25, что также означает, что она будет поддерживать весь функционал, введенный в предыдущих версиях.

Согласно официальной статистике, предоставляемой Google, распределение устройств между версиями приблизительно следующее:

Версия ОС	Кодовое имя	API level	Процент устройств
2.3.3 - 2.3.7	Gingerbread	10	0.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.9%
4.1.x	Jelly Bean	16	3.5%
4.2.x		17	5.1%
4.3		18	1.5%
4.4	KitKat	19	20.0%
5.0	Lollipop	21	9.0%
5.1		22	23.0%
6.0	Marshmallow	23	1.2%
7.0	Nougat	24	4.5%
7.1		25	0.4%

Таблица 1: Распределение версий среди устройств

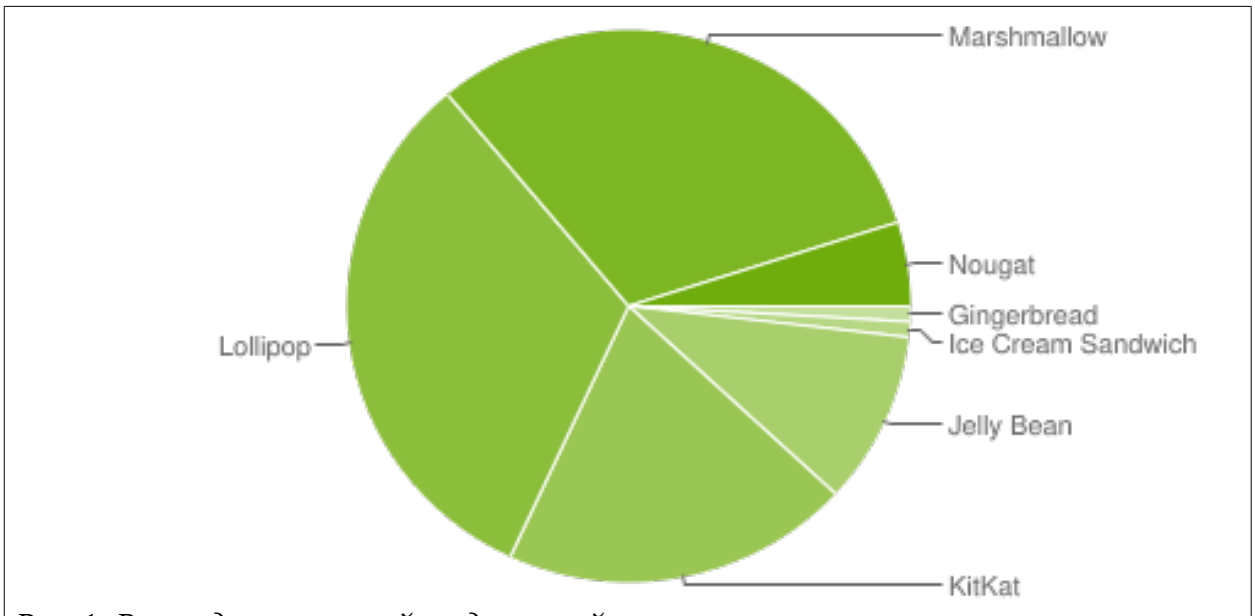

Рис. 1: Распределение версий среди устройств

Оптимальным уровнем, учитывая обхват устройств, а также требуемый функционал обеспечивает API level 16: при его выборе более чем на 95% всех устройств, имеющих доступ к официальному магазину Google Play Store, можно будет установить ArtQuest.

1.2 Макет activity.

Внешний вид приложения в любой момент времени определяет элемент класса, наследуемого системный класс `AppCompatActivity`. Для определения визуальной структуры пользовательского интерфейса существует макет, объявить который можно двумя способами:

1. Определить элементы пользовательского интерфейса в .xml, используя справочник xml-элементов для классов View и их подклассов.
2. Создать экземпляры элементов во время выполнения: создать и управлять свойствами объектов View и ViewGroup программно.

Имя элемента .xml для вида соответствует классу Android, к которому от относится. Так, элемент <TextView> создает виджет TextView в пользовательском интерфейсе, а элемент <LinearLayout> создает группу просмотра LinearLayout .

При работе с приложением возможно сочетать оба способа, к примеру — объявить в xml макет по умолчанию: элементы экрана и их свойства, а затем, с помощью кода в приложении — изменять в приложении состояние этих элементов, и создавать новые при необходимости.

Преимущество определения пользовательского интерфейса в файле макета в том, что так можно отделить представление приложения от кода, управляющего его поведением. Если описание интерфейса будет находится за пределами кода, то можно изменять или адаптировать его без необходимости вносить правки в исходный код и повторно компилировать его. Это также можно использовать для создания макетов экранов разных размеров, или различных ориентаций и языков.

Каждый элемент пользовательского интерфейса в приложении создаются на экране с поомщью объектов View и ViewGroup. Первый — формирует объект, с который пользователь может взаимодействовать, второй

— содержит другие элементы View и ViewGroup для опеределения макета интерфейса.

Android предоставляет коллекцию подклассов View и ViewGroup, включающую в себя как и обычные элементы ввода — кнопки, текстовые поля, переключатели, так и различные модели макетов — линейный, относительный, горизонтальный, абсолютный и т.д.

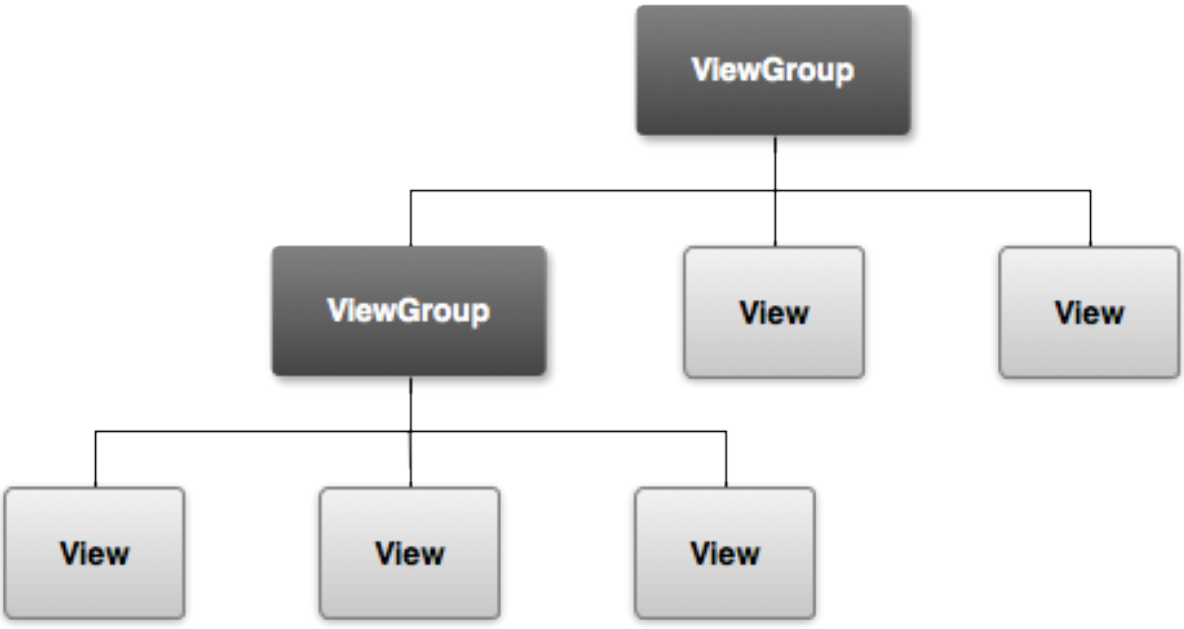

Рис. 2: Иерархия, определяющая макет изображения

Пользовательский интерфейс каждого компонента приложения определяется с помощью иерархии объектов View и ViewGroup, как показано на рисунке 2. Каждая группа — невидимый контейнер содержащий элементы ввода или другие виджеты.

Принципы дизайна интерфейса, отмеченные инженерами Google:

1. Simple vs Clear: интерфейс должен быть простым (не нагруженным) и понятным для использования

2. Content vs Chrome: необходимо использовать максимум экрана, при этом уменьшать его визуальную сложность (использовать ограниченное число кнопок/иконок)
3. Consistent yet engaging: консистентность реакции пользователя – пользователь должен понимать что он делает и как сделать то, что ему необходимо
4. Enhanced by cloud: данные пользователя следует хранить в облаке; пользователь должен иметь возможность выбирать настройки(организовывать данные) один раз, без повторных действий.

В течение работы приложения activity проходит через различные этапы своего существования, которые в общем называются activity lifecycle (рис. 1)

При этом, смена состояния вызывает определенный метод:

- onCreate() – при первом создании
- onStart() – перед тем, как activity будет видно пользователю
- onResume() – перед тем как будет доступно для взимодействия с пользователем
- onPause() – перед тем, как будет показано другая активность
- onStop() – когда activity становится не видна пользователю
- onDestroy() – перед тем, как будет уничтожена

При этом, необходимо учитывать, что поведение по умолчанию при повороте экрана подразумевает полное уничтожение существующей активности и создание новой.

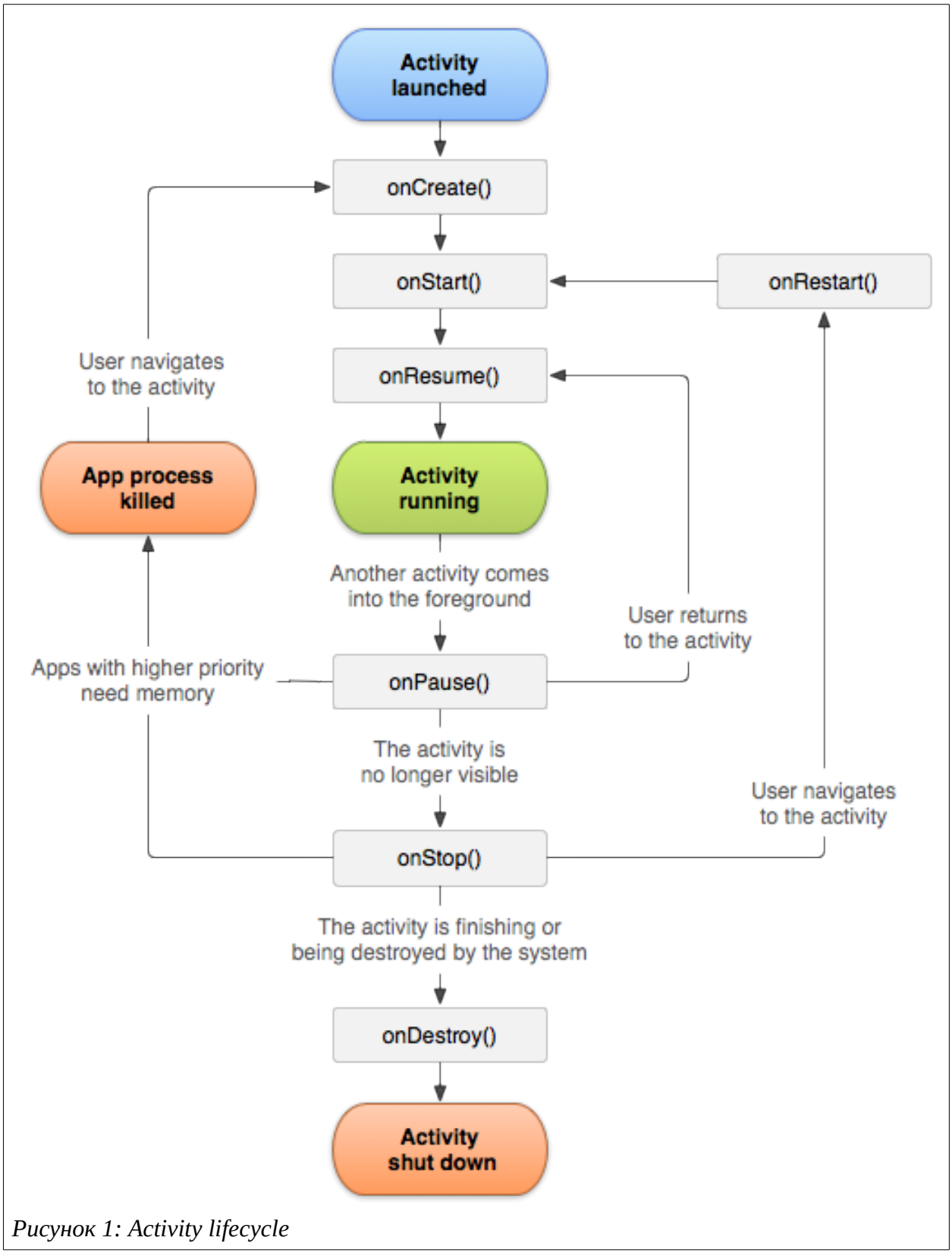

Рисунок 1: Activity lifecycle

1.3 Размеры элементов пользовательского интерфейса

Большое различие физических параметров устройств — разрешения экрана, его размеров и т. п. - приводит к тому, что необходимо реализовывать механизмы для одинакового отображения интерфейса, а также учитывать, чтобы игроку было удобно взаимодействовать с ним.

В 2006 году ряд исследователей университета Оулу, Финлянлия и университета Мэриленд объединились для выяснения оптимального размера кнопки для использования устройства одной рукой. Они протестировали два сценария: в первом пользователю необходимо совершить одиночное действие (запустить приложение, поставить отметку и т. п.), а во втором — цепочку таких действий (к примеру, ввести номер телефона). Для каждого сценария были использованы различные варианты размера кнопок.

В результате, процент ошибок значительно увеличивается, если диаметр кнопки для разового действия меньше 9.2 мм, для серии — 9.6 мм. При этом дальнейшее увеличение размера кнопок не ведет к значимому повышению точности нажатий.

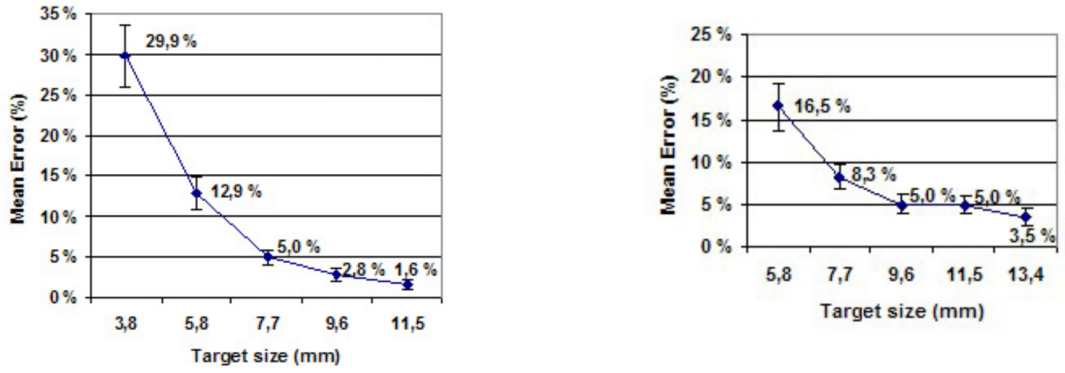

Figure 4. (a) Mean percentage of erroneous trials for each target size in discrete target study phase.

Рисунок 2: Средний процент ошибок для одиночных и серии нажатий

Пять лет спустя исследователи университетов Ольденбурга и Дуйсбурга-Эссена, Германия, повторили эксперимент. Их задачей было найти

оптимальный размер кнопок на сенсорном дисплее. Для своего эксперимента они создали игру на Android, с помощью которой зафиксировали более 120 миллионов касаний.

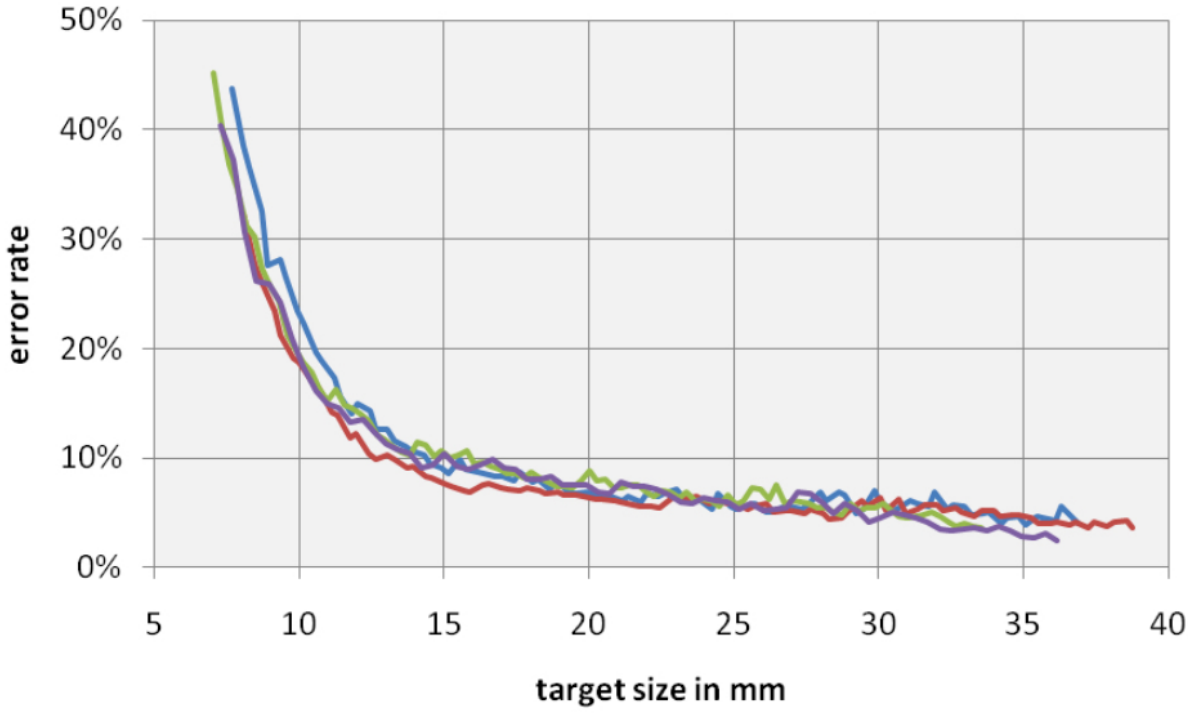

Рисунок 3: Зависимость количества ошибок от размера цели

Проанализировав все касания, были получены следующие выводы: на кругах с диаметром 15 мм и меньше количество промахов значительно увеличивалось. По мишеням размеров менее 8 мм игроки промахивались более чем в 40% случаев. И аналогично предыдущему исследованию, увеличение размера мишени более чем на 12 мм не ведёт к значительному величению точности попаданий.

Пол Фиттс, психолог из университета Огайо, в 1954 году разработал принцип, который станет известен как закон Фиттса, являющийся основой взаимодействия человека и компьютера. Закон Фиттса в общем случае — общий закон, касающийся сенсорно-моторных процессов, связывающий время движения с точностью движения и с расстоянием перемещения. Чем дальше или точнее выполняется движение, тем больше коррекции

необходимо для его выполнения, и соответственно, больше времени требуется для внесения этой коррекции.

Математическая запись закона:

$$T = a + b \cdot \log_2\left(\frac{D}{W} + 1\right)$$

Формула 1: Закон Фиттся

Где, T - среднее время затрачиваемое на совершения движения; a - среднее время запуска или остановки движения; b - величина, зависящая от типичной скорости движения; D - дистанция от точки старта до центра цели; W - ширина цели, измеренная вдоль оси движения.

Закон Фиттса применяется при проектировании графического интерфейса пользователя, позволяя определить размеры элементов интерфейса, их расположение и взаимное расположение на экране в соответствии с тем, насколько простым (или, наоборот, затруднённым) должно быть их использование.

Когнитивист и соавтор книги «Mind Hacks» Том Стэффорд очень точно суммирует важность логарифма: «Хотя суть этой формулы банальна (большие объекты легче выбрать), самое интересное в ней — точная математическая характеристика, а также то, что эта характеристика включает логарифмическую функцию, — это означает, что форма взаимосвязи между размером и временем реакции изогнута так, что даже небольшое увеличение размера для небольших объектов делает их гораздо более достижимыми (в то время как незначительное увеличение размера для больших объектов почти не повлияет на результат). То же самое относится и к изменениям расстояния до цели».

Использование результатов исследований и закона Фиттса позволяет создавать дружелюбные к пользователю интерфейсы.

1.4 Потоки

Если при отсутствии работающих запускается новый компонент приложения, система Android создаёт новый процесс для приложения с одним потоком выполнения. Поведение по умолчанию — работа всех компонентов одного приложения, в том числе только что созданных, в одном процессе и потоке, который называется главным. Этот поток отвечает за отправку событий на виджеты интерфейса, включая события графического представления. К примеру, методы, которые отвечают на обратные системные вызовы, такие как onKeyDown() для сообщения о действиях пользователя, всегда выполняются в главном потоке. Поэтому его называют потоком пользовательского интерфейса.

Если все события происходят в главном потоке, то выполнение долговременных операций — к примеру, сетевой доступ или запросы к базе данных, будет блокировать весь пользовательский интерфейс, и все события, включая изменения отображения, перестанут обрабатываться. С точки зрения пользователя приложения «зависает». И если в течение около 5 сек. Не произойдёт разблокирование потока — появится системное сообщение «приложение не отвечает».

Помимо этого, набор инструментов пользовательского интерфейса Andoid не является потокобезопасным. Таким образом, существует два правила однопоточной модели Android:

1. Не допускать блокирования главного потока
2. Не обращаться к инструментам пользовательского интерфейса Android снаружи предназначенного для этого потока.

Резюмируя всё вышесказанное, следует не допускать блокирования пользовательского интерфейса и выполнять долговременные операции в отдельных (фоновых) потоках.

Один из методов, позволяющих это сделать — AsyncTask. С его помощью можно выполнять асинхронную работу в пользовательском интерфейсе: он выполняет блокирование фонового потока и последующую публикацию результатов в потоке пользовательского интерфейса, без необходимости самостоятельно работать с потоками или обработчиками.

Метод AsyncTask позволяет выполнять асинхронную работу в пользовательском интерфейсе. Он выполняет операции блокирования в рабочем потоке и затем публикует результаты в потоке пользовательского интерфейса без необходимости самостоятельно обрабатывать потоки и/или обработчики.

Для использования этого метода необходимо создать подкласс AsyncTask и реализовать метод обратного вызова doInBackground(), который работает в пуле фоновых потоков. Чтобы обновить пользовательский интерфейс, используется метод onPostExecute(), который имеет доступ к результатам работа метода doInBackground() и работает в потоке пользовательского интерфейса — что даёт возможность безопасно обновлять пользовательский интерфейс. Задача выполняется через вызов метода execute() из потока пользовательского интерфейса.

Краткий обзор работы AsyncTask:

1. Можно указывать тип параметров, значения хода выполнения и конечное значение задания с помощью универсальных компонентов
2. Метод doInBackground() выполняется автоматически в фоновом потоке
3. Методы onPreExecute(), onPostExecute() и onProgressUpdate() запускаются в потоке пользовательского интерфейса
4. Значение, возвращенное методом doInBackground(), отправляется в метод onPostExecute()

5. Можно вызвать publishProgress() в любой момент в doInBackground() для выполнения onProgressUpdate() в потоке пользовательского интерфейса
6. Задание можно отменить в любой момент из любого потока

При этом, начиная с версии ОС Honeycomb все задачи asynctask при запуске кладутся в очередь и выполняются последовательно, хоть и в фоновом потоке. Данное поведение по умолчанию можно изменить, указав явно специальный ключ выполнения THREAD_POOL_EXECUTOR в при запуске задачи методом executeOnExecutor().

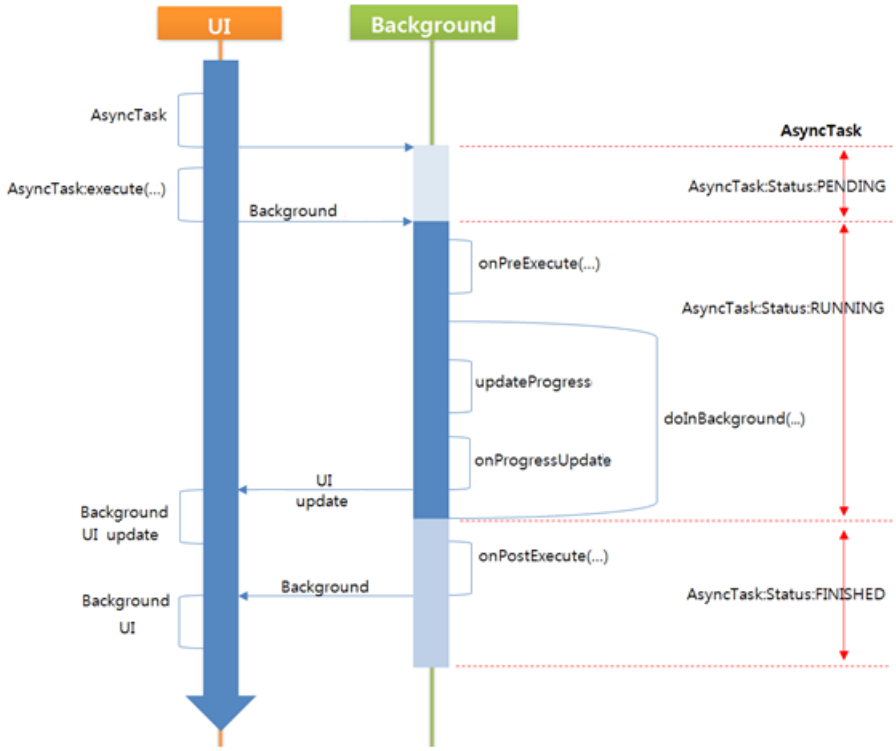

Рисунок 4: Схема выполнения AsyncTask

Однако, при использовании необходимо учитывать ряд особенностей и правил:

1. Задача должна определяться, создаваться и запускаться из главного потока. Это делается автоматически начиная с версии ОС Jelly Bean

2. Внутренние методы (onPreExecute, onPostExecute, doInBackfground и проч.) не должны запускаться вручную
3. Задача должна быть выполнена только один раз — при попытке запустить ту же задачу во второй раз будет ошибка.

Так, одна из проблем — непредсказуемый перезапуск метода при изменении конфигурации в процессе выполнения (например — изменение ориентации экрана), из-за чего возникает разрушение рабочего потока.

1.5 Хранение данных

Хранение данных в операционной системе android возможно следующими способами, каждый из которых обладает своими плюсами, минусами и специфики применения:

1. В базах данных SQL
2. В виде файла во внутренней или внешней памяти
3. В формате «ключ-значение»

1.5.1 Использование базы данных

Сохранение в БД идеально подходит для повторяющихся и структурированных данных, таких как контактная информация. API-интерфейсы, необходимые для использования базы данных на платформе Android, доступны в составе пакета android.database.sqlite. Например, полезный набор содержится в классе SQLiteOpenHelper. Он используется для получения ссылок на базу данных, при этом система выполняет потенциально долговременные операции создания и обновления базы данных только тогда, когда это необходимо, а не при запуске приложения. Для чего используются вызовы getWritableDatabase() или getReadableDatabase(). Следует учитывать возможную продолжительность выполнения, поэтому рекомендуется создавать для этих операций фоновый поток. Для использования SQLiteOpenHelper создаётся подкласс, заменяющий методы вызова onCreate(), onUpgrade() и onOpen() и, опционально - onDowngrade().

Для добавления строки в бд используется метод insert(), в который необходимо передать объект ContentValues. Для чтения — метод query() с передачей критериев выделения желаемых столбцов. Метод сочетает элементы insert() и update(), за исключением того, что список столбцов определяет данные, которые вы хотите получить, а не данные для вставки. Результаты запроса возвращаются в объекте Cursor.

Чтобы вызвать строку из базы данных в месте курсора используются специальные методы перемещения, которые необходимо вызывать перед считыванием значений. В большинстве ситуаций в первую очередь вызывается moveToFirst(), который помещает "позицию чтения" на первую запись в результатах. Для каждой строки значение столбца можно прочитать, вызвав один из методов Cursor get, например getString() или getLong(). Для каждого из методов get вы должны передать указатель желаемого столбца, который может вызвать getColumnIndex() или getColumnIndexOrThrow().

Для удаления строк из таблицы нужно указать критерии выделения, идентифицирующие строки. API-интерфейс базы данных обеспечивает механизм для создания этих критериев, предоставляющий защиту от внедрения SQL-кода. Механизм делит спецификацию выбора на предложение выбора и аргументы выбора. Предложение определяет столбцы для рассмотрения, а аргументы представляют собой значения для тестирования. Поскольку результат обрабатывается не как обычные выражения SQL, он защищен от внедрения SQL-кода.

Таким образом, класс SQLiteOpenHelper предоставляет удобный функционал работы с базой данных, избавляя пользователя от необходимости вручную писать sql код. В свою очередь, база данных — наиболее функциональный и производительный способ хранения большого объема структурированной информации.

1.5.2 Файлы

Операционная система Android использует файловую систему, похожую на дисковые файловые системы других платформ. Объект File подходит для последовательного чтения или записи больших объемов данных — изображений или любых других файлов, передаваемых по сети.

Все устройства Android имеют две области хранения файлов: внутренняя память и внешние хранилища. Эти области появились в первые годы существования Android, когда на большинстве устройств имелись встроенная память (внутреннее хранилище) и карты памяти (например micro SD, внешнее хранилище). Некоторые устройства делят встроенную память на внутренний и внешний разделы, так что даже без съемных носителей в системе две области хранения файлов, и API-интерфейс работает одинаково вне зависимости от типа внешнего хранилища.

Внутренняя память	**Внешнее хранилище**
Всегда доступна.	Доступно не всегда: пользователь может в любой момент подключать и отключать такие хранилища
Файлы доступны только приложению	Файлы доступны для чтения везде
При удалении приложения система Android стирает из памяти все относящиеся к нему файлы	При удалении приложения файлы стираются только если директория их сохранения была указана с помощью метода getExternalFilesDir()

Таблица 2: Сравнение особенностей внутренней и внешней памяти

Таким образом, внутреннее хранилище используется когда необходима уверенность, что ни пользователь ни другие приложения не получат доступ к файлам, а внешнее — для файлов без ограничений доступа.

1.5.3 Формат ключ-значение

Если необходимо сохранить относительно небольшой набор пар "ключ-значение", используется API-интерфейс SharedPreferences. Объект SharedPreferences указывает на файл, содержащий пары "ключ-значение", и предоставляет простые методы для чтения и записи. Управление каждым таким объектом осуществляется с помощью инфраструктуры и может быть

частным или общим. Так, частные настройки будут удалены после закрытия приложения, а общие — сохранятся вне зависимости от его статуса.

Чаще всего используются для хранения настроек текущей сессии, либо глобальных настроек приложения. Не подходит для хранения большого объема данных.

Глава 2. Основная программная часть приложения ArtQuest

2.1 Модель приложения

Структура работы приложения можно представить в виде следующих шагов:

1. Определение пользователем настроек игровой сессии: количество вопросов и их сложность
2. Загрузить вопросы (вопросы):
    1. На основе полученных данных сформировать и отправить на сервер запрос вопроса (вопросов)
    2. На сервере, используя алгоритм подбора на основе рейтинга пользователя и картины выбрать из БД вопросы, подходящие под критерий сложности
    3. Из списка вопросов к случайно выбранному добавить три случайных варианта ответа и преобразовать в строку JSON для выдачи в ответ на запрос
    4. На устройстве, полученную строку распарсить, полученные данные внести в локальную БД
    5. Среди полученных данных выделить адрес изображения и загрузить его на внутреннюю память для дальнейшего использования
    6. Повторять шаги 2-6 до тех пор, пока не будут загружены все вопросы
3. Параллельно с загрузкой:
    1. По запросу от пользователя выводить конкретный вопрос из локальной БД на экран устройства

2. После того, как игрок дал ответ (кликнув на вариант ответа) — запросить с сервера информацию о его правильности
3. На сервере произвести пересчёт рейтинга автора, картины и пользователя в соответствии с правильностью данного ответа

Таким образом, необходимо как минимум две активности для обеспечения функционала:

1. Стартовое меню, с информацией о выбранной сложности, количестве вопросов, и кнопкой, начинающей новую игровую сессию с данными настройками
2. Активность — шаблон, для наполнения данными о вопросе из БД, предоставляющая следующую информацию:
    1. Текущий номер вопроса
    2. Количество загруженных вопросов на данный момент
    3. Картину, с возможностью масштабировать
    4. Четыре варианта ответа
    5. Дополнительные кнопки для навигации

И надо учитывать разнородность хранимых данных и обеспечить к ним доступ вне зависимости от состояния приложения и устройства:

1. Текстовые данные о вопросе хранить в локальной базе данных
2. Загруженную картину — во внутренней памяти
3. Данные о настройках игровой сессии — при помощи механизма задания глобальных настроек приложения (формат ключ-значение)
4. Прочие, необходимые для работы приложения переменные хранить вне активностей, чтобы их значение хранилось вне зависимости от activity lifecycle)

2.2 Рейтинговая система

Для определения уровня сложности картины используется система рейтинга, основанная на шахматной.

Каждому автору, пользователю и картине на сервере присвоен собственный рейтинг, по умолчанию равный 1000. При ответе на вопрос происходит учёт всех трёх значений, в результате чего, при правильном ответе — рейтинг игрока растёт, а картины и автора — падают, при неправильном — соответственно наоборот.

При запросе нового вопроса происходит сортировка всех картин по рейтингу, и первая треть считается сложной, вторая — нормальной, а третья — простой сложности. Таким образом, пользователи сами влияют на то, какие картины входят в ту или иную категорию.

2.3 Основной функционал приложения

Для того, чтобы избежать ошибок связанных с зависимостью asynctask от состояния текущей активности, а также был доступ к ряду данных вне activity lifecycle при запуске приложения создаётся элемент класса, названного GameToolset, все методы которого обеспечивают нужный функционал. Для автозапуска и независимого существования этот метод прописан в android manifest file.

Список методов и переменных класса GameToolset:

- Методы для: начала новой сессии; отмены активных задач; ответа на вопрос; инициализации нового вопроса;

```
public void StartNewSession(Context context) {...}
public void CancelSession() {...}
public void AnswerTheQuestion (Integer QuestionNum, Integer answer_Num)
{...}
public void InitTaskActivity (final Activity current_activity, Integer
QuestionNum) {...}
```

- Переменные: сложность сессии; количество вопросов; порядковый номер последнего загруженного вопроса; UID — уникальный идентификатор пользователя на сервере; ID, с которого начинается текущая сессия в БД; переменная для доступа к текущей активности и изменения её содержания

```
private Integer diff;
private Integer amount;
private Integer current;
private String UID;
private Integer startID;
private ActiveActivity activity;
```

- Классы

- Asynctask: `taskDownloadQuestionChain` – последовательная загрузка, обработка всех вопросов в сессии, добавление информации в БД

```
private class taskDownloadQuestionChain extends AsyncTask<Void, Void, Void>
{…}

taskDownloadQuestionChain current_taskDownloadQuestionChain;
```

- Asynctask: `taskAnswerTheQuestion` – загрузка информации о правильности ответа, обновление строк в БД

```
private class taskAnswerTheQuestion extends AsyncTask<Integer, Void, Void>
{...}

taskAnswerTheQuestion current_taskAnswerTheQuestion;
```

- Asynctask: `taskInitTaskActivity` – проверка, загружены ли текущие вопросы (есть ли информация в БД и сохранено ли изображение в памяти). В тот момент, когда они будут загружены — вывод информации на экран

```
private class taskInitTaskActivity extends AsyncTask<Integer, Void, Void>
{...}

taskInitTaskActivity current_taskInitTaskActivity;
```

- DBHelper: `DBHelper` – обеспечение доступа к БД, контроль её версий, создание при установке приложения и проч. Также отдельно инициализированы три соединения

```
private class DBHelper extends SQLiteOpenHelper {...}
DBHelper db_AddData = new DBHelper(this);
DBHelper db_UpdateData = new DBHelper(this);
DBHelper db_InitData = new DBHelper(this);
```

- `ActiveActivity` – класс для более простого доступа к текущей активности.

```
private Activity activity;
private ProgressBar pbar;
private Button[] author = new Button[4];
public ActiveActivity (Activity current_activity) {...}
public void setPbarPrimaryProgress(Integer progress, Integer amount) {...}
public void setPbarSecondaryProgress(Integer progress) {...}
public void setAuthorNames(String[] author_name) {...}
public void setAuthorColors(Integer answer_Num, Integer answer_RightNum)
{...}
public void setImage(String URL) {...}
public void setVisibility(Boolean isLast) {...}
```

2.4 Схема многопоточности

Для обеспечения корректной работы приложения и доступа к локальной базе данных без блокирования главного потока необходимо реализовать многопоточность. Было решено использовать asynctask, учитывая специфические особенности использования. Так, для того, чтобы метод работал вне зависимости от состояния главного потока и текущей activity необходимо обеспечить доступ к методам вне цикла активностей — для этого они объявлены в классе GameToolset.

Как упоминалось ранее, всего используется три задачи: для запроса одного вопроса с сервера, его обработки, и добавления информации в базу данных; для проверки правильности ответов и для проверки наличия данных в БД и памяти.

2.4.1 Метод `taskAnswerTheQuestion`

Определяется и вызывается в методе `StartNewSession(Context context)`, который, как следует из названия, отвечает за начало новой игровой сессии. В нём происходит очистка данных с прошлой игры, завершение всех активных фоновых потоков, которые могли остаться с прошлой сессии, начальная инициализация, используя настройки из локального хранилища и создание задачи на загрузку вопроса.

В методе doInBackground этой задачи происходит:

1. Установка соединения с сервером
2. Загрузка JSON строки
3. Её парсинг
4. Добавление новой строки в БД, соответствующей загруженному вопросу.

В методе onPostExecute, который выполняется в главном потоке строго после фоновой задачи, выполняются следующие операции:

1. обновление индикатора, отвечающего за отображение количества загруженных вопросов (при помощи обращения к методу объекта класса ActiveActivity, который определяется при начале нового вопроса)
2. Создание задачи по загрузке изображения
3. Создание задачи по запросу следующего вопроса с сервера

Таким образом, если задача не была отменена — происходит последовательная загрузка всех вопросов текущей игровой сессии в фоновом потоке.

2.4.2 Метод taskAnswerTheQuestion

Определяется и вызывается в методе `AnswerTheQuestion (Integer QuestionNum, Integer answer_Num)`, который запускается после того, как пользователь выберет вариант ответа нажатием на соответствующую кнопку. В метод передаётся порядковый номер текущего вопроса (хранится непосредственно в activity) и порядковый номер выбранного ответа (соответствует нажатой кнопке).

В методе doInBackground:

1. Прежде всего, происходит переход от порядкового номера вопроса (от 1 до n, где n – выбранное пользователем максимальное количество вопросов в сессии) к ID вопроса в базе данных
2. Затем, используя полученный ID из БД запрашивается информация о вопросе, для формирования запроса на сервер: идентификаторы картины и вариантов ответа

3. Для окончательного создания запроса необходимо перейти от порядкового номера ответа (1...4) к идентификаторам, полученным из базы. После чего, с сервера запрашивается очередная JSON строка, с информацией о правильности данного ответа.
4. Полученная строка парсится
5. Затем с помощью полученных данных обновляется строка в БД, соответствующая данному вопросу. Вносится информация о правильности, порядковых номерах корректного и данного пользователем ответов

   В onPostExecute:

1. При помощи обращения к методам элемента класса ActiveActivity изменяется внешний вид активности. Правильный ответ подсвечивается зелёным, неправильный (если не совпадают) — красным. Все остальные удаляются.
2. Становится активна кнопка перехода к следующему вопросу в сессии

Так как данная задача должна выполняться параллельно загрузке вопросов, при запуске используется ключ THREAD_POOL_EXECUTOR. Если этого не делать — проверка правильности ответа будет проведена только после того, как все вопросы в сессии будут загружены.

2.4.3 Метод taskInitTaskActivity

Определяется и вызывается в методе `InitTaskActivity (final Activity current_activity, Integer QuestionNum)`, который вызывается при создании новой активности с вопросом. В него передаются переменные, соответствующему порядковому номеру вопроса и ссылка на только что созданный объект класса activity — она нужна для определения

объекта класса ActiveActivity и последующего изменения внешнего вида активности используя методы данного класса.

В методе doInBackground:

1. Переход от порядкового номера к ID в БД
2. Происходит циклическая проверка доступности информации о вопросе в базе (существует ли строка с заданным ID)
3. После успешного прохождения проверяется, есть ли в памяти загруженное изображения, соответствующее этому вопросу

При успешном прохождении всех проверок запускается метод onPostExecute, в котором происходит запрос соответствующих вопросу данных, и, используя методы класса ActiveActivity — вывод информации на дисплей.

Так-же как и с ответом на вопрос, если выполнять эту задачу без ключа THREAD_POOL_EXECUTOR — она не будет завершена до тех пор, пока не будут загружены все вопросы.

2.4.4 Загрузка и сохранение изображений

Для параллельной загрузки изображений, сохранения их в локальную память для последующего доступа — кеширование — было решено использовать стороннюю библиотеку, обеспечивающую более высокую производительность, в сравнении со встроенными методами.

Использованы следующие настройки библиотеки Universal Image Loader:

1. Неограниченное кэширование на физической памяти
2. Название файла соответствует таковому на сервере
3. Для загрузки используются три дополнительных потока

Их достаточно всего один раз задать при запуске приложения.

Загрузка изображения, проверка доступности, очистка кэша реализованы используя встроенные методы этой библиотеки.

2.4.5 Заключение

Таким образом, при работе приложения могут быть активны следующие потоки:

- Главный
- Последовательная загрузка всех вопросов
- 3 потока для загрузки изображений
- Для проверки ответа
- Для проверки доступности данных

2.5 Хранение данных

При работе с базой данных необходимо учитывать особенности, к примеру, не допускать одновременного обновления одних и тех же строк, контролировать соединения и проч. Так, на каждое активное соединение может в момент времени приходиться только одна операция чтения / записи, поэтому таких соединений необходимо открывать несколько.

Так, в течение работы программы может быть активно одновременно три соединения к БД:

1. Добавление новой строки с только что загруженными данными
2. Обновление информации в соответствии с данным ответом
3. Запрос данных для отображения нового вопроса

Также, ряд данных необходимо хранить вне зависимости от состояния приложения — настройки текущей игровой сессии. Для этого используется механизм «ключ-значение» для всего приложения. Таким образом хранятся три значения: UID пользователя, представляющий собой string строку с системным идентификатором устройства, параметры diff и amount — соответственно сложность и количество вопросов в сессии. Пользователь может изменить последние два параметра, вызвав всплывающее меню с главного окна.

Как упоминалось выше, для загрузки и хранения изображения используется сторонняя библиотека Universal Image Loader. При этом, в качестве настроек локального кэша выбран «без ограничений», так как согласно тестам, проведенным автором библиотеки, это обеспечивает максимальную производительность. Однако, для избежания переполнения памяти на устройстве кэш автоматически очищается при запуске новой игровой сессии.

2.6 Жесты и интерфейс

При повседневном использовании смартфона многие привыкли к определенным жестам — например смахивание влево — вправо для перелистывания. Именно поэтому действие некоторых кнопок было продублировано подобными жестами:

1. Во время игровой сессии можно смахиванием перелистывать вопросы, возвращаясь к уже решенным
2. Изображение картины можно приблизить, что может помочь определить автора, если получится разглядеть подпись
3. Повернув дисплей (сменив ориентацию) можно развернуть картину на весь экран

При построении интерфейса выдержан минималистичный стиль, использованы только стандартные элементы, что ускоряет отрисовку экрана.

Рисунок 5: Скриншот интерфейса: первый вопрос

Рисунок 6: Скриншот интерфейса: правильный ответ

Рисунок 7: Скриншот интерфейса: неправильный ответ

Рисунок 8: Макет активности

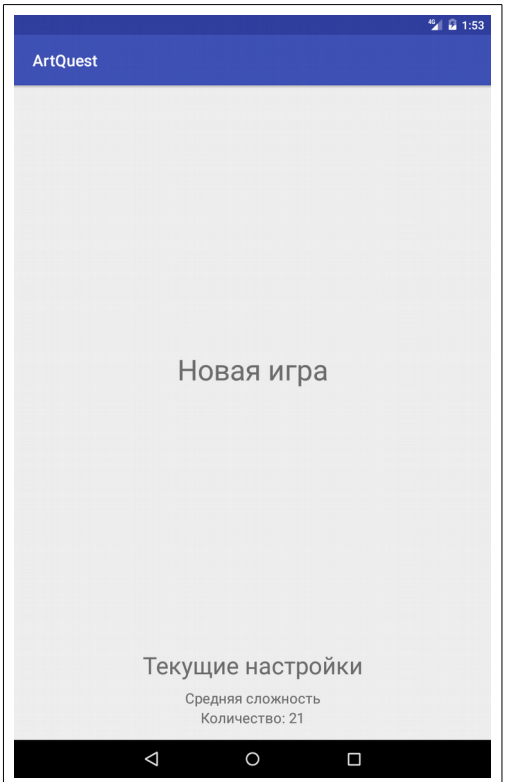

Рисунок 9: Скриншот интерфейса: главное меню

Выводы

Были изучены многочисленные аспекты полного цикла производства приложения для операционной системы Android, в том числе особенности использования многопоточности, хранения данных, запросов информации с сервера и её последующая обработка.

Полученные знания использованы для создания приложения. Для обеспечения функционала и подготовки приложения к релизу был проведён ряд тестов на физических и виртуальных устройствах.

Заключение

В следующих обновлениях планируется реализовать следующий функционал:

1. Система достижений: награды выдаются за правильное выполнение определенного количества вопросов заданной сложности
2. Отображение статистики: график изменения рейтинга со временем, какого автора игрок знает лучше всего и т. п.
3. Сохранение и передача профиля игрока на другое устройство
4. Оптимизация подбора вариантов ответа: сбор статистики по каждой картине, с целью определить кого, помимо автора картины, наиболее часто за него принимают. Использование полученных данных для усложнения вопросов

Список литературы

Статья в журнале

1. 100,000,000 Taps: Analysis and Improvement of Touch Performance in the Large, 2011
2. Target Size Study for One-Handed Thumb Use on Small Touchscreen Devices, 2006
3. I Would Not Plant Apple Trees If the World Will Be Wiped: Analyzing Hundreds of Millions of Behavioral Records of Players During an MMORPG Beta Test, 2017
4. Play and Learn: Using Video Games to Train Computer Vision Models, 2016

Ссылка на документы в интернете

1. Android Developers **https://developer.android.com/index.html**